# Assessment of Software Quality with Static Source Code Analysis: GridSpace2 Case Study

Bartłomiej Bodziechowski[1], Eryk Ciepiela[2], Marian Bubak[1,2]

[1] AGH University of Science and Technology, Department of Computer Science AGH,
al. Mickiewicza 30, 30-059 Kraków, Poland
[2] AGH University of Science and Technology, ACC Cyfronet AGH,
Nawojki 11, 30-950, Kraków, Poland
emails: bubak@agh.edu.pl,e.ciepiela@cyfronet.pl

**Keywords**: software engineering, software quality, static source code analysis, GridSpace

## 1. Static code analysis

Testing and source code analysis methods are the pillars of assuring quality of software. While the former is nowadays not contested at all, the latter is often not paid due attention to, neglected or even ignored during software development process. Static source code analysis can significantly augment software engineering methods not only in assuring software quality considered as *conformance with requirements* or *usability*, but, even more importantly, when dealing with rapidly changing requirements, evolving goals, inconstant environment, and dynamic set of functionalities included. In this aspect, new categories of software architecture quality and its deficiency symptoms such as e.g. rigidity, fragility, immobility, viscosity and illegibility need to be constantly investigated on the earliest possible stage of software development process [1]. Static source code analysis offer methods for identifying and measuring such symptoms in an automated fashion that can be incorporated into software development product-line.

## 2. Methodology applied

The presented research started with a thorough survey of the state-of-the-art in software architecture design principles in general, and specifically - methods dedicated to object-oriented paradigm, and tools implementing these methods for software written in the Java programming language. The methods and tools presented define and discuss a suite of metrics – functions that map units of software to values that are interpreted as the degree of fulfillment of a certain property of the quality of software.

Afterwards, the authors carry out a case study of GridSpace2 Virtual Laboratory [2] – a distributed computing platform aimed at scientific computations that enables researchers to author, run, share and publish so called *virtual experiments* over Grid-based resources and other computing infrastructures. This kind of software is subjected to continuous change for several reasons, e.g. specific requirements of a variety of already existing and new distributed programming approaches have to be embraced, multiple computing back-ends are to be supported, e-infrastructures evolve, as well as services and software packages installed on e-infrastructure change. Therefore, GridSpace2 architecture quality was investigated using selected state-of-the-art static source code analysis methods and tools that were applied against GridSpace2 code.

## 3. Results of the GridSpace2 software analysis

The values of computed metrics were obtained and included in a comprehensive report along with a recommendation on possible improvements to be applied to the code [3]. A number of architecture, design and implementation rules and good practices violations was

discovered, with one of them being depicted in Figure 1. Moreover, it was shown how the developed tools can be incorporated into daily software development process.

## 4. Conclusions and future work

Static source code analysis methods and metrics proved very useful in verification of software design. Although they cannot replace design experts in creating good architecture, yet still they can answer whether an architecture meets the desired characteristics of well-designed system. It's worth noticing that such analysis, owing to already existing tools, can be automated thus it does not increase costs of software development. The tools presented in this work are easy to be enabled and incorporated into continuous software development process. According to periodically generated software quality report, GridSpace2 will be subjected of refactoring on a regular basis.

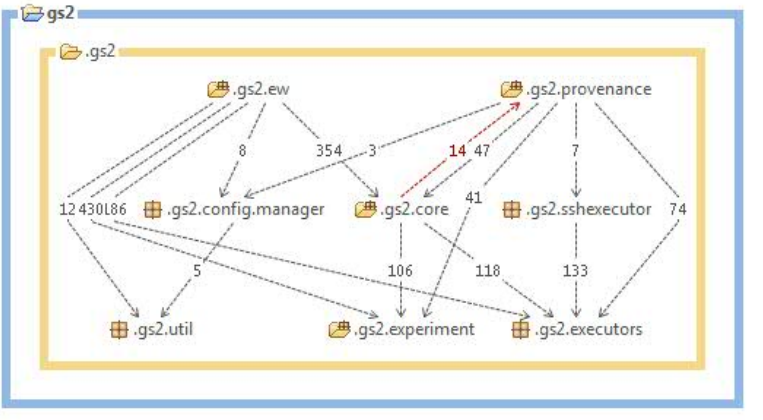


Fig. 1. Overview of GridSpace2 packages analyzed by Stan tool [4] in terms of inter-package references. Arrows show dependencies between packages, arrow labels denote a number of references between packages. Obtained graph should form a directed acyclic graph, yet one loop was detected.

**Acknowledgements:** This research was partially funded by the EU IST MAPPER project. The authors are very grateful to Marek Kasztelnik for fruitful discussions and suggestions.

## References

1. R. C. Martin: Design Principles and Design Patterns, `http://www.objectmentor.com/resources/articles/Principles_and_Patterns.pdf`, 2000.
2. E. Ciepiela, P. Nowakowski, J. Kocot, D. Harężlak, T. Gubala, J. Meizner, M. Kasztelnik, T. Bartynski, M. Malawski, M. Bubak: Managing Entire Lifecycles of e-Science Applications in the GridSpace2 Virtual Laboratory - From Motivation through Idea to Operable Web-Accessible Environment Built on a Top of PL-Grid e-Infrastructure. In: M. Bubak, T. Szepieniec, K. Wiatr (Eds) Building a National Distributed e-Infrastructure - PL-Grid - Scientific and Technical Achievements, Springer 2012, ISBN 978-3-642-28266-9, pp. 228-239 (2012).
3. B. Bodziechowski: "Analysis of Software Quality Using Metrics as Applied to Given Examples", MSc Thesis, AGH University of Science and Technology, 2012, MSc Thesis, AGH Krakow, September 2012, available at `http://dice.cyfronet.pl`
4. Stan tool – Structure Analysis for Java, `http://stan4j.com/papers/stan-whitepaper.pdf`